VR

Definition of distributed Computing by Streit:

DC is about hw & sw systems that work together at a geographical distance acting as a problem solving environment for the benefit of their users

Examples: Web services / virtualization
Grid Computing
Big Data (distributed Big Data & analysis like map reduce)
Cloud Computing

Metacomputing: aggregation of multiple, geographically distributed supercomputers to solve problems too large for each supercomputer respectively (re-entry of spacecraft, ..)

Grid concept:
new class of computing infrastructure based on the internet supposed to provide scalable, secure and high-performant mechanisms for resource discovery, access negotiation, use of remote resources
=> enables scientific collaborations ("virtual organizations")

Grid Checklist:
coordinates resources that are not subject to centralized control
uses standard, open, general purpose protocols (for access, discovery, ...)
delivers non-trivial QoS (response time, throughput, security, co-allocation)
the utility of the combined system must be greater than the sum of its parts

Grid, like a power grid. But! Access control, more services, not standardized

Cloud Computing
ubiquitous, convenient, on-demand, shared pool of resources, rapidly provisioned, minimal-to-no management effort, scalable, flexible, fully-automatic, pay-as-you-go service

Distributed Systems & Middleware

A distributed system is a collection of <u>independent computers</u> that appear to its users as a single <u>coherent system</u>

Interaction between individual servers & users hidden & consistent

Should be scalable, extensible & fault tolerant

A grid infrastructure can be considered as a highly distributed system offering non-trivial QoS

Goals: 1. Connecting users & resources
2. Transparency (often trade-off decisions)
3. Openness (offer access using (open source) standards)
4. Scalability (allow growth in resources, users, administrative domains)

1. Sharing HW/SW and services
 often economical reasons (expensive supercomputers)
 simplifies collaboration & exchange (groupware)
 requires security mechanisms
 Billing & accounting

2. Hide HW differences on different machines (byte-order, architecture, CPUs,) & their location by using URL
 assure consistency
 Failure handling: distinguish between dead & slow resources
 hide failure & recovery

3. Protocols specify format, content, order & meaning of messages
 Interfaces ": syntax used to access certain resources / functions
 Semantics hard to formalize, often informal descripted // comment
 Interoperability: use other's interfaces for co-existence
 Portability: run in various systems (tablet, phone, TV, ..)
 Flexibility: easy configuration out of different components from potentially different developpers

4. Avoid main bottleneck (single server) by spreading resources

new problems:

Interconnect become new bottlenecks, communication, consitency, unreliable communication, no efficient broadcasts possible, asynchronous in WANs, location service needed

independent administrative domains involved

no pre-exiting trust

requires mutual protection => increased system complexity

use certificates

Web Services (=> Web resource)

Middleware technology based on XML & the web

Language & platform neutral for machine-to-mahine communication

accesible via HTTP, provides API (Web-API)

in a machine-readable format called from another program

described in a WSDL document (Web Service Description Language)

typically registered and can be accessed w/ URL

XML Message exchange can be done using SOAP

Web Services are provided by web servers to the WWW

using URLs & well-known ports

Clients access compute power, storage, services using the above

Web Services are not for users, they are designed to be behind the scenes. the client software invoke Web service calls

+ loosely coupled
+ platform independent
+ language "
+ self-descibing
+ co-existance w/ firewalls (HTTP / HTTPS / SMTP (...)
+ separation of interface definitions from their embodiment
+ simplicity & extensibility
- Data conversion overhead (C / C++ / Java ⟷ XML)

Web services are a connection of technology
=> form the basis for modern grids

WSDL
describes syntax of messages for invocation & response of WS
is an XML Document -> self-describing web service
consists of 3 fundamental parts:
1. what the service does => its functions w/ arguements & returns
2. How to access: details on data formats & protocols
3. Where it is: protocol-specific network access (e.g. URLs)
WSDL is not used during communication
SOAP / HTTP / HTTPS is used
WS-provider publish their WSDL files, costumer uses it
provide interface for programmers, machine & human readable

XML (Extensible Markup Language)
Meta-Markup language for text documents
defines generic syntax to markup text using tags
only general structure for tags, not the tags themselves

WSRF (Web Service Resource Framework)
Service interfaces often imply the need for some form of stateful interaction between client & server
one operation may influence the following ones
XML document describes state of the resource
5 Specifications of WSRF
1. WS-Resource (id)
2. WS-Resource-Properties (data fields, methods, ...)
3. WS-Resource-Lifetime (destroy resources after time)
4. WS-Resource-Group (collection of resources)
5. WS-BaseFaults (set of information displayed in case of fault)

Security

Grid communication uses open internet

Intruders may read/alter/collect/flood messages

Requirements:

Identification of individuals (eg using private keys) & passwords

nice to have: single sign in: use resources multiple times

use standards like X.509, agree on a PKI

Interoperability w/ local security solutions

users may have different names @ different sites or

receive a dynamic guest account, use Kerberos or SSL

Support for multiple implementations

In case of certificates: always need for a central trust-

worthy entity that checks personal identity in real life before

handing out a certificate

=> certification authority (CA)

Certificates contain public key as an attribute

authentification done by a signing challenge

PKI (Public Key Infrastructure)

High demand => single CA would be overloaded => chain of trust

2. Models: 1. higher CAs sign only CAs, that sign users

2. CA certificates Registration Authorities (RAs)

that can sign users

CA issues certificates for servers/clients on request

has Certificate Revocation List (CRL)

answers issues concerning certification issues

define policy for users and their conditions

RA is an office of CA that accepts requests for certificates

checks identity of a person

does not issue own certificates

Drawbacks of X509
high provisioning overhead
require a lot of knowledge from the user
private keys must be properly secured
Idea: use credentials that exist @ home institution (eg KIT e-mail address) to issue short lived certificates that can be hidden from the user => Shibboleth

Job Submission
Job ≙ some work to be done, typically requires CPU time / RAM or access more different resources
Resources: CPU cores / RAM / nodes } computation
Data (size, throughput, transfer rate, ...)
Network (bandwidth, latency, ...)

Differentiation of Computer Systems
by #users, usage (super-computers, servers), #cores

Cluster Systems
Large unified computers running multiple user accounts, jobs & tasks that typically demand more resources than available in size (#cores) and time (> cycle)
=> scheduler needed
Scheduler uses policies to come to a schedule:
fairness / efficiency / minimized response time / turn-around time / maximized throughput / utilization

Each submission system differs in commands, policy, options, underlying hardware architecture, requirement specification format (file / command / ...)
Jobs themselves differ: nodes / task, #tasks, shapes
=> abstraction desired: JDSL

JDSL (Job Submission Description Language)
serves for interoperability
does not define a submission interface / the results of a submission or their shape / how resources are selected
JDSL code can not be directly submitted into Grid resources
Steps to run a grid job from the perspective of a grid system:
1. Resource discovery
2. Resource selection and allocation
3. Execution & job management
1. Determine a suitable set of resources for the job
   query resource availability & status & used policies
   based on the published information by the resource providers
   Authorization filtering: check if necessary access rights exist
   Minimum requirement filtering: if multiple resources
   can be allocated, choose the one w/ lowest overpower
  Problem: discovery may not have revealed all needed information
2. Dynamic information gathering
   use published scheduling to estimate completion time
   Assign resources to the job
   Verify if match is still valid (if not, repeat steps before)
   Allocate resources finally
3. Translate job into grid-specific job and poll its status
   Handle & stage job data: initialization & destruction
   In case of multi-step jobs: control & monitor workflow
Classification of grids
  Checklist again: 1. not subject to centralized control
                   2. standard, open, general purpose protocols
                   3. non-trivial QoS
Classification: By resource type shared / technology used

Compute Grids / Data Grids / Peer-to-Peer Grids / Collaborative

Compute Grids: access to computational resources
- Desktop Grid: ex. SETI @ home
- Server Grids: Linux systems
- HPC / Cluster: high-end supercomputers

Data Grids: transparent, secure, high-performance access to data across adminstrative domains
- Flat file / relational / streaming data
- For sharing data between researchers

P2P grids: sharing storage capacity of desktop PCs beyond LAN
- bitTorrent, GNUtella, Napster

Collaborative grids: central resource: human expertise
- human interaction across geographical distance & organizational boundaries

Classification by organization / geographical range
- Departmental / Enterprise / Global / Utility / Service

Anatomy defines grid as protocol stack (hourglass model)
- Small set of core abstractions to allow diverse application to be mapped to diverse set of resources
- Includes: accessible services, APIs, SDKs

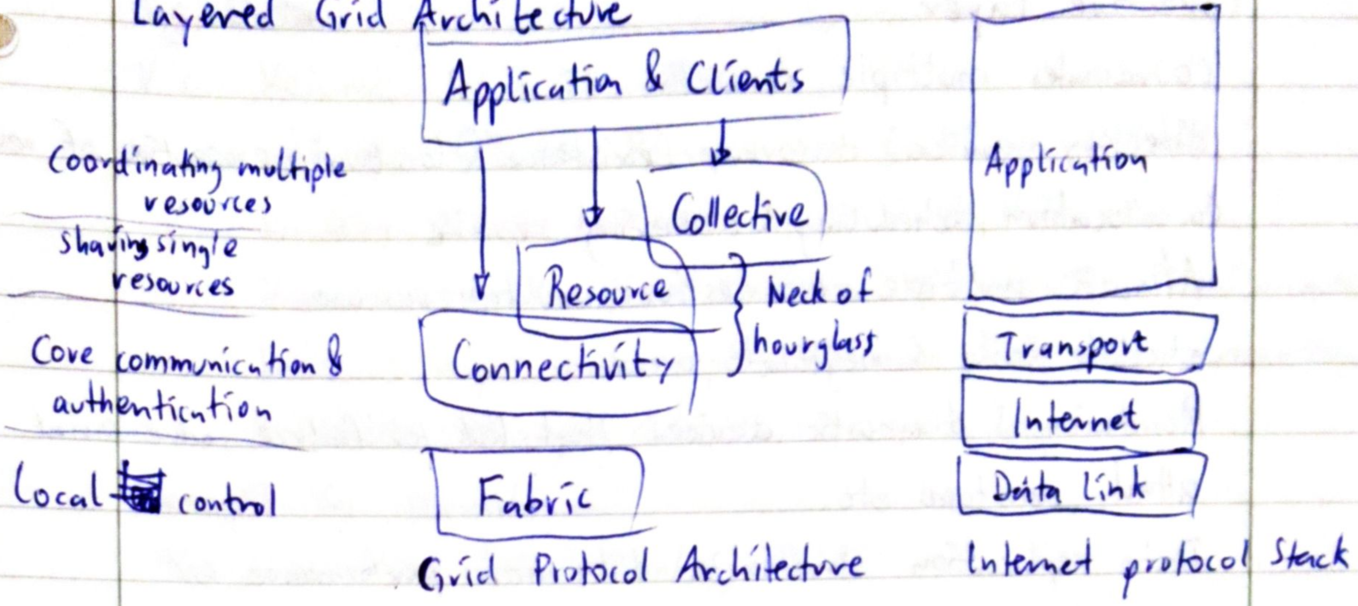
Protocol $\hat{=}$ Convention / standard enabling communication between 2 end points (protocol implementation may vary)

Service $\hat{=}$ Capability accessed via protocol / remote interface

API $\hat{=}$ description how to access a set of functions

SDK $\hat{=}$ set of development tools allowing creation of applications for a given software package, implements API

Layered Grid Architecture



Fabric layer provides resources to be shared
offers local, HW-specific operations: enquiry mechanisms, available space, load/queue state, starting/monitoring/reservation/put/get = operations

Connectivity layer: enable exchange of data between fabrics
Transport / Routing / Naming
Authentication Protocols provide secure mechanisms for verifying identities & resources
Allow access to multiple resources w/ single sign-in
Allow program to act on user's behalf (delegation)
Integration w/ local security solution (UNIX / Kerberos)
Establish trust relationships

Resource Layer: sharing of single resource
Defines standard protocol for the secure negotiation, initiation, monitoring, control, accounting & payment of sharing operations
Implementations call fabric-layer functions
Information protocols: state & structure of a resource
Management protocols: access, operations, accounting, payment, monitoring & control

Collective layer
coordinates multiple resources
directory service: discovery (existence & location) & properties of resources
Co-allocation, scheduling & brokering service
Allocate multiple resources for simultaneous use
Schedule tasks & negotiate access
Monitoring & diagnostic devices that look for failure, adversarial attack, overload etc..
Data replication: maximize data-access performance w/ respect to response time, reliability & cost

Physiology of the grid
Defined in a standardized framework, Open Grid Service Arch. (OGSA)
Service oriented: all resources presented as a service
all components in the environment are virtual
=> Aligns grid technology w/ web services
extends notion of a web service to address stateful behavior
=> explicit distinction between a service & stateful resource
Includes a set of basic interfaces to access state of a resource
Use of WSRF to define infrastructure for integrating & managing services within a virtual organization
Execution management services
Data / resource management / Security / self-management or information services
OGSA is only a framework for the structure of grids & the organization of services
First detailed specification of OGSA service model & interaction in Open Grid Service Infrastructure (OGSI)
„Too" object oriented, works not well w/ existing web services and too overloaded for one standardization
Led to WSRF

Big Data

V's: Volume (GB, TB, EB, ...)

Velocity (Batch, Periodic, near Real Time, RT)

Variety (data base, Web, Audio, Photo, Video, Social ...)

Veracity (Systematic & statistical uncertainties, biased sample)

Value (unique, non-reproducible data, high cost of reproduction)

Variability (changing data, linkage, changing models)

Example for veracity

US News Law School Rankings 2015

used for reputation & expectation => high fees $50k

Non-manipulatable factors Σ 42,75%

Strong tendency for manipulation: Σ 11,25%

Highly manipulatable factors: Σ 46%

Data Management in LHC:

L1 specialized HW → online farm → online farm → WW Physics community

Tiered Structure: Tier-0: Data Aquisition, highest availability, data distribution to Tier-1

Tier-1: online to data acquisition process, grid-enabled service, tape storage, data-intensive analysis, high availability

Tier-2: Less data intensive tasks, lower requirement on availability, often run by physics institutes / universities

Data Analysis

Reconstruction of physical reactions, time consuming, standard processes & parameters, often run @ Tier-1, centrally organized

Monte Carlo simulation of events, usually for specific physical processes, often run @ Tier-2

User analysis w/ specific parameters, low efficiency on raw

data => coupled in analysis, usually @ Tier-2 / Tier-3
Data Management: Sites run storage elements
File access via grid commands
Logical File Catalog map LFN to PFN using GUID
Problem: dark data
Data Stewardship
Responsible use and protection of digital assets through management, infrastructure support & sustainable practices
Tasks: Analyzing data for quality & reconciling issues
creating & maintaining consistent metadata
defining the roles & responsibilities of the people involved
Lead government practices, in particular endorse good data management practices, use them and share them
Handling data ownership
ensuring data preservation
Facilitating data curation
Data Preservation: Actions taken to ensure long-term viability and availability of digital assets
ensuring materials'
Authenticity, Reliability, Usability, Integrity
Long-term: depends on the data, up to "forever"
challenge: new data formats, storage interfaces, users
Bit-stream preservation: physical preservation of data objects on the 'bit-level'
use redundant storage, heterogenous, standardized technology, replication to remote sites, refreshment of hardware, exchange to newer technologies (if any), security & disaster planning

Economical aspects of data preservation

What to preserve?

Which materials are essential? Primary? What Software?

How to preserve? Metadata, formats? Media, storage? Data center?

Paying the costs? Who pays? Business model? $\frac{1}{2}$ ingest $\frac{1}{3}$ preservation % access

How long? What institutions will exist then? In 100 yrs?

Ingest & preservation need 2 be paid upfront. Access on demand

Free-rider problem: A pays for ingest & preservation, B benefits of it

Who has access? What tools used?

Who is responsible? Data creators? Professional communities? Government?

How should preservation be done? Curation / Metadata / Storage / Facilities / Monitoring / Migration / Copies?

Data Curation: Maintaining & adding value to a trusted body of digital information for current & future use.

Typically user initiated & maintains metadata rather than the database itself

Aspects of use: 1. Accessibility (authentication)

2. digital rights management

Aspects of value: 1. Annotations

2. Linking to related data

3. Improving & updating documentation

Open Access

Classical scientific publishing

1. Authors submit an article to a conference / journal
2. Peer-review
3. Publication of reviewed article in conference / journal
4. Research institutions subscribe to journals / publishers to allow access to articles

(Used to be having actual copies on site, nowadays online, pricey!)

Green open access (free?)
Publish work & self-archive manuscript version
Gold open access (€100 -€X000)
Publish & immediately available & freely available via the publisher website
Hybrid open access
Business model: relies on subscriptions / fees
Gold access for publishers' articles available on specifically paid fee
Double dipping ≙ double payment for publishers
Subscription fees + article processing charge
PIDs: persistent identifier, long-lasting reference to a document / file / web page or other object
Idea: enable clicking on it
PIDs only work, when resolved to a real address
PID resolving done by a system responsible for forwarding users to the location referenced to, this service needs to be known to the user
Setting up PID service is easy but:
organizational issues: Business model, rights, neutrality, exit strategy
Infrastructure & security: location independence, migration possibilities, scalability, technology independence, back-up system
Objects: validity, uniqueness, transparence, straightforwardness
Metadata: informational data that describes data
Administrative: acquisition / location information
documentation of legal access requirements
Descriptive: Title / abstract / date of creation / creator / Annotations / keywords

Technical: Formats / compression / scaling routine / authentication / encryption keys / passwords

Preservation: documentation of physical condition of resources / " of actions taken for preservation

Use: use & user ~~ranking~~ tracking, content re-use

Dublin Core Metadata Element Set

Small set of core vocabulary terms to describe web resources & physical resources

15 classic metadata terms:
- Title
- Creator
- Subject / Topic
- Description
- Publisher
- Contributor
- Date
- Type
- Format
- Identifier
- Source
- Language
- Relation
- Coverage (the spatial / temporal applicability under which the source is relevant)
- Rights

Map Reduce

Massive Data can not be stored on one machine.
Takes too long to process in serial.
⇒ Need for a programming model for processing large scale data w/ automatic parallelization, friendliness to procedural languages, scalibility to petabytes on thousands of machines

Basic value type: key-value pair (k, v)

Map step: apply operation to data → Reducer step: merge same intermediate values

map: $(k, v) \mapsto ((k_1, v_1), (k_2, v_2), \dots (k_n, v_n))$

reduce: $(k', (v_1', v_2', v_3', \dots v_n')) \mapsto (k', v'')$

mappers transform input records into intermediate records

reducers reduce a set of intermediate values, which share a key, to a smaller set of values

Map Reduce interfaces

Mapper + Reducer

Data Split ⇒ Input Files ⇒ Mappers ⇒ Intermediate Results
⇒ Suffle ⇒ Reducers ⇒ Outputs ⇒ Combiner ⇒ Output

Page Rank

used to find "best" pages matching a query
using link structure

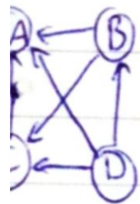
Several formulations:

Voting system: rank of page is based on the weighted votes of all pages linking to that site

Eigenvector - Problem:
$Ar = r$ where r is the vector containing the ranks of all pages and A is the adjacency matrix

Random - Walk / Markov Chains:
If a surfer randomly surfes the web, the page rank expresses the probability of the surfer to access a given page @ given time

Example:



The (simplified) algorithm:

$\sum PR_s \in (0,1)$

$$PR(u) = \frac{1-\beta}{N} + \beta \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

$\beta \hat{=}$ damping factor (usually 0,8)
Set B_u contains all pages linking u
$L(v)$: # outgoing links from v

Map emits: (A, B/2), (C, B/2)
(A, C)
(A, D/3), (B, D/3), (C, B/3)

Reduce receives: (A, [B/2, C, D/3)
(B, D/3)
(C, B/2, B/3)

Cloud Computing

Traditional IT service provisioning: 1 App per OS per HW

What if one needs more HW for one app?

Or need to provide multiple Apps?

Cloud / Virtualization: 1 big HW, 1 virtualization / cloud layer & on top: many OS + many Apps

Expectation: reduction of costs for operators (through consolidation)

" " " " users (through pay-per-use)

reduced complexity for " (through higher usability, lower time 2 solution)

easier access (through web)

flexibility & user satisfaction (through virtualization)

"unlimited" resource pool

Increased freedom (through OS flexibility)

⇒ another shift of platforms, will change the way people think & work w/ HW

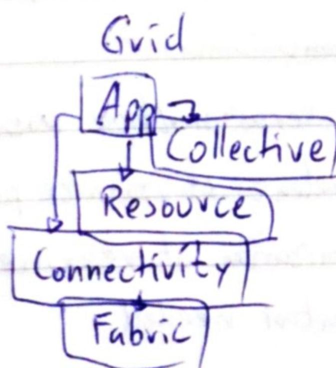
Cloud Computing: ubiquitous, convenient, on-demand access to an "unlimited" pool of resources (network, storage, computation power, applications, servers, ...) that can be rapidly provisioned with very little or no overhead / management

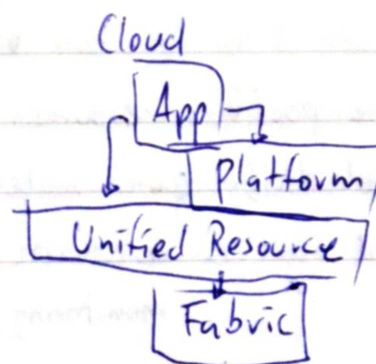
Cloud fails the Grid checklist:

a cloud usually is controlled from a central point

a cloud does not use open & standardized access protocols, it uses proprietary protocols

often deliver non-trivial services, but not necessarily





	Grid	Cloud
Objective	sharing of resources	making money
Infrastructure	many distributed sites	few datacenters
Central control	no	yes
enablich technology	HPC, middleware, OGSA, WSRF	virtualization, Web 2.0
Middleware	standard based	propietary
user interface	various, APIs, command line	browser based
used by	academica	academica & industry
business model	use for free	pay-per-use
funding	public	commercial

Why is cloud computing so attractive?
access via browser (easy, nice)
available when needed
on-demand resource provision
customized environment
self-service w/o administrator
cost-effective

NIST: 5 essential characteristics
On demand self service: customer can access computing capabilities as needed w/o requiring human interation
Broad access via network: capabilities are available over network & accessed using standard mechanisms from various platforms
Resource pooling: resources are pooled to serve multiple users
Rapid elasticity: quick scale-out & rapid release of resources provisioning
Measured service: automatic control & optimization of resources monitoring, reporting & control needed

NIST: 4 Deployment models

Private Cloud: operated solely for one organization
owned, managed & operated by that "itself
or 3rd party
On or Off premises

Community Cloud: Supports a specific community that shares concerns
Owned, managed & operated by the community
or a third party
On or Off premises

Public Cloud: Available to the general public
Owned, managed & operated by business, academic or
government, combinations of these possible
On premises of the cloud-provider

Hybrid Cloud: Composition of > 1 of the above
Parts remain unique entities but are bound together
to enable data & application portability

NIST 3 Service Models

Iaas Infrastructure as a service
Features: Provisioning of fundamental computing resources
Consumers are able to deploy and run arbitrary software
Underlying cloud architecture is transparent, no
need for management or control

Platform as a service PaaS
Features: Allows consumers to develop their applications in the cloud
Programming environment & tools are supported by the provider
Consumers do not manage / control underlying infrastructure
Applications are normally presented as Web Services

SaaS Software as a service
Features: Allow consumers to use the provider's software running on the provider's architecture
Licensed software
Applications of public interest
Accessible through a thin client interface such as a Web browser
Related data is stored remotely
consumers do not manage / control the underlying infrastructure (network, server, OS, storage, computing resources)

XaaS Everything as a service
Storage as a service
Use the storage of the service provider
Cost-effective (pay-per-stored-size / pay-per-transferred)
Grid as a service
HPC as a service

What is a service?
Low barriers to entry: making them available to small business
Large scalability
Multitenancy (allow resource sharing)
Device Independence (allow users to access the system on different HW)

Underlying Technology
Web Service: Enables info from one app to be made available to others
" internal apps to be made available over the internet
" an easy access via existing tools & mechanisms
Virtualization: Enables customized environments & on-demand resource provisioning
Virtual Machines (VMs) rather than physical ones
An addition layer on the physical machine (hypervisor)
Starting an instance of VM via VM image
⇒ Web service + virtualization $\hat{=}$ cloud computing

Virtualization

Cloud computing deploys virtual machines to provide on-demand resources
=> Running multiple OSs on the same hardware
=> Need for hypervisor architecture or hosted architecture

Hypervisor: directly installed on the bare metal, direct HW access, very efficient, great scalability & robustness, limited to specific HW configurations

Hosted architecture: Virtualization on top of OS, supports broad range of HW configurations, less efficient & scalable

Advantages of virtualization:
+ On-demand OS / resource customization
+ Performance isolation (VMs isolated from each other)
+ Security (attacks should only compromise one VM)
+ Availability (easy migration of VMs)
+ Management (done by every VM individually)
+ Legacy support
+ Access "root" privilege

capex: capital expenditures
opex: operation expenditures

	Virtualization	Cloud
Definition	Technology	Methodology
Purpose	create n environments on 1 HW	deliver variable resources
Configuration	Image based	template based
Lifespan	Years, long term	hours - months, short term
Cost	↑ capex, ↓ opex	private: ↑ capex, ↓ opex, public ↓ capex ↑ opex
Scalability	Scale up	scale out
Workout	stateful	stateless
Tenancy	single tenant	multiple tenants

Hypervisor (Virtual Machine Monitor VMM)

serves for virtualization for CPU / RAM / IO / GPU ...

takes care of VM scheduling & resource allocation

CPU (x86) virtualization

x86 OSs are designed to run in ring-0, they assume to own the HW (there are ring 0 - ring 3 for hardware access management)

=> virtualization layer must be under ring-0

Full virtualization

translates kernel code to replace non-virtualizable instructions w/ new sequences of instructions that have the intended effect on the virtual HW

Full abstraction (complete decoupling) from the underlying HW

=> guest OS is not aware of running in a VM

VMM translates all OS instructions on the fly (& caches results for future use). User space instructions run unmodified @native speed

Best isolation & security for VMs, simplifies migration & portability for guest OSs & applications

Paravirtualization

Modifies OS kernel to replace non-virtualizable instructions w/ hypercalls that communicate w/ virtualization layer hypervisor

also for memory management / interrupt handling / time keeping

OS is aware of running in a VM, poor compatibility & portability

HW Assisted virtualization

new CPU execution mode in HW, allowing VMM to run below ring-0

=> No binary translation. No hypercalls

Memory Virtualization

Map the memory of a VM to the shared physical memory

Virtualization of the MMU & TLB

VVM uses TLB HW to map the virtual memory directly to the physical memory => avoidance of two-level-translation on every access

Device & I/O virtualization

Hypervisor virtualizes the physical HW & presents each VM a standardized set of virtual devices

done by simple device abstraction or by real HW emulation

Manages the routing of I/O requests between the virtual devices & the standard physical HW

VMware uses full virtualization

KVM (Linux kernel module for VMs)

Xen: Paravirtualization

Virtual Appliance (why is a VM not enough?)

VA is a virtual machine image w/ pre-installed software (OS + software)

Eliminates installation, configuration, maintainance costs

used for SaaS for providers

~~Performance evaluation~~

Virtualization on top of the OS (Containers)

because it is not always needed to have many heterogeneous OSs on the same HW

VMs in user space, use the same kernel & existing OS mechanisms for isolation purposes

Container: complete runtime environment of the application including libraries & config files

lightweight, close to bare metal performance

VM - OS ≙ Container

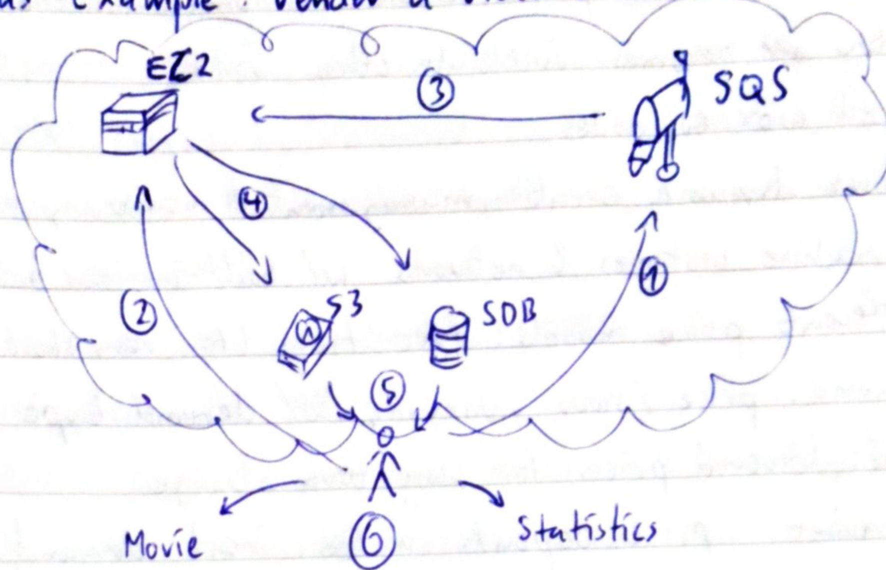
Docker: Virtualization 2.0

Docker containers wrap up: code, runtime, system tools, libraries, ... in one file system built on Linux Containers (LXC)

add portability in app bundles, are app centric, optimized, offer versioning, component re-use via base-imaging, can be shared in public registry

Cloud Examples

IaaS Example: render a video in the cloud using AWS



SQS: simple message queue — SDB: shared database

EC2: ExeCute 2 — S3: scaling object storage

(1) creation of one subtask for each video frame

(2) creation of n worker nodes

(3) fetch tasks from queue & start procession

(4) generate frames & statistical data

(5) retrieve data

(6) compose video & evaluate performance / statistical data

Steps to take

1. Job enqueue
2. worker node allocation
3. setup central storage & metadata database
4. Automate job acquisition
5. prepare template for worker nodes containing required software

Simple Queuing Service

allows user to push tasks, can be dequeued out of order

If a worker node receives a task, this task is not deleted

immediately from the queue. After some time t of worker load and no completion, failure is assumed and the queue entry becomes visible to other workers

Creation of worker nodes

EC2 allows dynamic creation, management & monitoring of virtual machine instances & networks w/ built-in load balancing

Four different price models: free tier (for very short period)

- on-demand: price / hour, varying w/ demand type
- reserved: lowered prices for long-term usage
- spot market: price depends on current demand
  - works by placing a bet: as long as the price is lower as the maximum bid: bidder receives HW
  - Spot-market instance may be cancelled by Amazon

For monitoring purposes, a database for metadata is necessary

SDB: scaling, relational db, accessible via SQL-like syntax

For the output frames, use S3 (SSS = Simple Storage Service)

a distributed object storage for data, stored in buckets

1B - 5TB / object can be stored

integrated into Amazon's content delivery network

BitTorrent support

data can be flagged private / public

Move "cold" data to Amazo Glacier

support for user defined attributes

Logging, versioning & encryption on server side

Job automation

a worker item should automatically start executing tasks from the job queue. Done by a simple script upon boot

Creating a template: EC2 instances base on Amazon Machine Images (AMAs) => choose one from Amazon/

third party, or creat an own one
on boot an script is exewted, ssh keys copied over,
adjustments for personal need possible
Install rendering software on an Ubuntu (for example)
and store this machine Image as template for all workers
PaaS example
No access to Infrastructure / OS
Only access to API
Heroku supports pragramming languages: Python, Ruby, Java, PHP, NodeJS
Developer locally develop code which can be deployed via
CLI or git where it is executed by "dynos".
Dyno = flexible computing unit
Data is stored in DBs because no FS is accessible
Heroku runtime orchestrates dynos
Application access via Browser (HTTP)
1. Specify application dependencies (libraries, versions, ...)
2. Write code (minimal running example first !!)
3. Specifying start command w/ "procfile"
4. Deploy the application
1. & 3. lead to platform specific metadata
=> can become very complex
in addition, limitations due to the distributed & managed
character of platform applications apply
SaaS example
Instead of providing infrastructure / development platform:
provide end-user ready application in the web
web messenger / data warehouse / groupware system
Software can easily be customized to match the enterprise's
corporate identity & can be integrated in the internal network
of the enterprise

Well known examples:
Google Docs
Office 365
Adobe Creative Cloud
=> user can benefit from not having to care about maintenance & of always up-to-date applications
Pricing usually on monthly basis
Cloud infrastructures are capable & flexible enough to enable SaaS on a global scale

Cloud Components
Cloud: an elastic, virtual, on-demand, "infinite", always accessible solution for various concerns
scale up / down: in- / decrease the performance of resources
scale in / out: in- / decrease the amount of resources

CAP Theorem
Consistency: all components are in the same state
Availability: all components reply to requests as expected
Partition tolerance: the system as a whole works as expected even with single components failing
Examples: DNS: A, P Relational Database: C, A
Banking applications: C, P

Multi-Tier technology
Single Process
Client → Server
Client → Server → Data
Client → Server → Service → Data
enables load distribution, amount / type of layers not limited
Limited change propagation, well scaling, but longer deployment time

low Tier architectures suffer from hard code maintenance of monolithic applications & large, expensive nodes
high-tier architectures suffer from complex communication & large administrative overheads due to in-homogeneous environments

P2P architectures

Services provided by peers / nodes in a single big network
Peers are treated w/ no central organization or hierarchy
Super-Peers act as mediators between P2P networks

+ no central component
+ single client + server application on al nodes
+ resource distribution among all nodes
+ high scalability
- complex message routing
- security issues (errorous client deployed everywhere)
- hard to backup
- Peers have to fulfil more than one role (↑ load, ↑ complexity)

Service oriented Architecture (SOA)

Atomic services provide a single functionality, applications are implemented composing these
Load distributed among several different, specialized services
standardized interfaces for communication & reusability
control mechanisms restrict access

+ modular approach: isolated entities allow flexibel scale-outs on demand
+ complexity & management overhead for each service low
+ allow for different communication mechanisms
+ metering per service possible
- flexible interfaces can be complex ⇒ slow deployment
- in sum high management overhead
- possible security issues if services are shared

Elastic computing resources
virtual CPUs (vCPUs) incorporated in VMs
Virtualization concept allow dynamic resource distribution
\# of vPUs as well as \#cycles per vCPU adjustable
Hypervisor measures resource usage w/o affecting performance
Virtually no limit of vCPUs
⇒ maximum flexibility for users, maximum administrative overhead for operators
VMs bundle CPUs, memory, storage, network → templating
Basic cost models (usually time based)
np-hard: selection of optimal \# VMs
Elastic platforms (JVM, .NET, CLR)
running on elastic infrastructure
operated by provider to reduce overhead for users
usually (only) pay-per-use
per-user configuration
Specialized computing resources
Elastic map reduce / video encoding / batch pipelines ..
cost models differ, depending on the resources used
user can save money, time & effort, if used correctly
Elastic Storage
on block level: reached by scaling logical address space
blocks usually distributed for maximum performance
on file level: managed by provider
Allow existence of metadata
Protocols: FTP, NFS, SMB, ...
single namespace for all files
storage can be scaled up to achieve performance/capacity boost
File based locking as well as replication possible

Key-value-stores

Storage for data objects (metadata + data)

addressing done by user-defined key

distributed back-end storage

keys are hashed to find the server that holds value

access control on object level

no locking, no folders "flat namespace"

highly redundant, replication inherent

access via HTTP

very easy to scale out: no underlying FS / DB, data location derived from key

(key, value) → (hash(key), value) → replication

Problem w/ replication: consistency

writing replicas may fail

strategy: when reading / writing: wait for certain amount (>1) of nodes to succeed

wait for 1 node to succeed

wait for all nodes to succeed

Strong consistency given, when:

$R + W > N$

R := # nodes reading from

W := # nodes writing to

N := # replicas

*DC = Data Centre

	Block Storage	File storage	Object storage
Transaction units	Blocks	Files	Objects
Protocols	SCSI, FC	SMB, FTP, NFS	HTTP
Metadata support	NO	YES (FS)	user-defined
Best suited for	Frequently changing	shared file data	static data
Biggest strength	Performance	Access & management	scalability, distributed access
Limitation	Scalability	Scalability beyond DC*	Frequent data changing
Scalability	to data centre level	to data centre level	global level

Virtual Networking
Infrastructure level communication
Network interfaces can either be emulated in Software or
virtualized by a hypervisor
SDN & VLAN tagging used for network isolation & flow control
Higher-Level protocols have to be implemented
=> Fast!

Message based mechanisms
→ direct messaging, i.e. remote procedure calls (RPC)
→ message queuing i.e. advanced message queing protocol (AMQP)
→ event driven messaging, i.e. enterprise service bus (ESB)
→ publish & subscribe mechanisms
Different message queues depending on use case
FIFO / Stack / Ring buffer
All methods have to cope w/ same aspects
Addressing & Routing
Message formatting
Delivery
Message based architectures: enable asynchronous app design
allow parallel processing of messages
increase robustness & availability of the system
decouple client & server
delay execution of messages

HPC / Parallelrechner
shared memory multi core computers
communication implicit via shared variables
2 models: one monolithical shared memory: UMA
every core w/ own memory, but can be
accessed from any core via interconnect: NUMA

message-oriented multicore architecture
communication explicit through messag passing
example: meta-computer
Scalable, but programs need to be more coarse-grained
Metric for performance: FLOP/S
does not scale necessarily w/ frequency of core
GPUs may calculate thousands FLOP/clock cycle
before x86 became common, specialized HW was used for GP
=> COTS-Cluster became popular (commercial-off-the-shelf)
Every node in a cluster computer runs its own Linux & contains itself a multi-core architecture
hybrid form of NUMA of UMAs (two Layers of parallelism)
$T(1)$: execution time on 1 core
$T(p)$: execution time on p cores
$S(p)$: speedup $\frac{T(1)}{T(p)}$
$E(p)$: efficiency $\frac{S(p)}{p}$

Amdah's law: let q be the portion of a program that can be parallelized
=> Total execution time:

$$T(p) = \underbrace{T(1) \cdot \frac{q}{p}}_{\text{parallel}} + \underbrace{T(1) \cdot (1-q)}_{\text{sequential}}$$

$$\Rightarrow S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{T(1) \cdot \frac{q}{p} + T(1)(1-q)} = \frac{1}{\frac{q}{p} + (1-q)}$$

$$\Rightarrow \boxed{S(p) \leq \frac{1}{1-q}}$$

Ideally, $S(p) = p$ most of the time: $1 \leq S(p) \leq p$
But there might also be super-linear speedup $S(p) > p$!
This might be due to larger caches on a SMP, which leads to less cache misses => time save => speedup